International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 4 Issue: 2 162 - 166

Comparison and Enhancement of Sorting Algorithms

Neha Gupta
Department of Information Technology
K.J Somaiya College of Engineering,
Mumbai, Maharashtra.
nehargupta05@gmail.com

Abstract— Some of the primordial issues in computer science is searching, arranging and ordering a list of items or information. Sorting is an important data structure operation, which makes these daunting tasks very easy and helps in searching and arranging the information. A lot of sorting algorithms has been developed to enhance and aggrandize the performance in terms of computational complexity, efficiency, memory, space, speed and other factors. Although there are an enormous number of sorting algorithms, searching and sorting problem has attracted a great deal of research and experimentation; because efficient sorting is important to optimize the use of other algorithms. This paper is an attempt to compare the performance of seven already existing sorting algorithm named as Bubble sort, Merge sort, Quick sort, Heap sort, Insertion sort, Shell sort, Selection sort and to provide an enhancement to these sorting algorithms to make the sorting through these algorithms better. In many cases this enhancement was found faster than the existing algorithms available.

Keywords- Sorting, Complexity, Efficiency, Bubble sort, Selection Sort, Insertion Sort, Heap sort, Quick Sort, Merge Sort, Shell Sort.

*****

I. INTRODUCTION

Sorting algorithm is one of the most elementary research fields in computer science. Sort is an essential operation in computer programming. In computer programming, a sorting algorithm is an efficient algorithm which performs a cardinal task that puts elements of a list or array in a certain order or arranges a collection of items into a particular pattern or order. There is a huge number of sorting algorithms in computer science and they differ from each other on the basis of efficiency of the algorithm. Sorting algorithms are usually adjudicated by their efficiency. In this case, efficiency refers to the algorithmic efficiency that is the behavior of the algorithm as the size of the input grows large. Most of the algorithms in use have an algorithmic efficiency of either O(n^2) or O(n*log(n)).

The processing time of a sorting algorithm is based on the two important factors which involve processing speed of a Processor and the internal memory (RAM) used by the system. The two types of sorting algorithms with respect to their algorithmic efficiency are O(n^2) (which comprises of the bubble, selection, insertion, cocktail, gnome and shell sorts) and O(n log n) (which includes the heap, merge, and quick sort). Efficiency of an algorithm depends on some major factors like CPU (time) usage, Computational Complexity, speed, Memory usage, Disk usage, Network usage. Most naïve sorting algorithms require two steps to sort data which includes comparing two items followed by swapping or copying. This process continues to execute over and over until all the data or elements are sorted. Sorting algorithms are classified in two categories according to the place whether they are stored, that is, in the main memory or auxiliary memory. One category is the internal sort which stores the data elements in the main memory and another is the external sort which stores the data in the hard disk. In fact, we can convert external sort algorithms to internal sort by utilizing the splitting and merging.

II. CRITERIA FOR COMPARISON

Many algorithms that have the same efficiency do not necessarily have the same speed and behavior on the same input. First and the most important factor is, algorithms must be judged based on their best case, average case and worst case efficiency There are some algorithms that show different behavior for different sets of input. Algorithms, such as quick sort, perform exceptionally well for some inputs, but horribly for others. And there are other algorithms, such as merge sort, are unaffected by the order of input data.

The most crucial factor is the “big-O notation”. Sorting algorithms are sometimes characterized by big O notation in terms of the performances that the algorithms capitulate and the amount of time that the algorithms take (where n is integer). Big O notation describes the limiting behavior of a function when the argument tends towards a particular value or infinity, usually in terms of simpler functions. Big O notation allows its users to simplify functions in order to concentrate on their growth rates. The different cases that are popular in sorting algorithms are: - O(n) is fair, the graph is increasing in the smooth path. - O(n^2): this is inefficient because if we input the larger data the graph shows the significant increase. It means that the larger the data the longer it will take. - O(n log n): this is considered as efficient, because it shows the slower pace increase in the graph as we increase the size of array or data [6].

The second criterion is stability. A sorting algorithm is considered to be stable if two objects with equal keys appear

in the same order in the sorted output as they appear in the input list of elements to be sorted. Some sorting algorithms are stable by nature like Bubble Sort, Insertion sort, Merge Sort, etc. Most of the simple sorting algorithms preserve the order of keys with equal values but advance algorithms like heap sort, do not.

The third criterion for judging algorithms is their space requirement. It is decided on the basis that do they require obliterate space or can the list of items be sorted in place (without the need of additional memory except a few variables)? Some algorithms never require extra space such that only O(1) or O(log n) memory is needed beyond the items being sorted, whereas some depends totally on external space requirement (it is important to create auxiliary locations to store data temporarily) and they are most easily understood when implemented with extra space (heap sort, for instance, can be done in place, but conceptually it is much easier to think of a separate heap). Space requirements may even rely on the data structure used (for instance, merge sort on arrays versus merge sort on linked lists) [9].

III. SUMMARIES OF POPULAR SORTING ALGORITHMS

A. Bubble Sort

Bubble sort is the simplest and unfortunately the worst sorting algorithm (keeping computational complexity in mind). It makes the use of a sorting method called exchange method. The name comes from the fact that each element "bubbles" up to its own proper position. This sort will do double pass on the data set of elements and swap two values when necessary. So, if we have 10 elements then the total number of comparisons is 100. The algorithm continues the process for each pair of adjacent elements until the end of the data set. This leads to the algorithm being extremely slow. This algorithm's average and worst case performance is O(n^2). So, Bubble sort is rarely used to sort large and unordered data sets. This causes larger values to "bubble" to the end of the list while smaller values "sink" towards the beginning of the list. Bubble sort can be used to sort a small number of items (where its inefficiency is not a high penalty) [10].

Bubble sort has many of the same properties as insertion sort, but it has slightly higher overhead as compared to insertion sort. In the case of nearly sorted data, bubble sort takes O(n) time, but it requires at least two passes through the dataset whereas, insertion sort requires only more one pass. The main advantage of Bubble Sort algorithm is the simplicity and ease of implementation of the algorithm. Another advantage is that the space complexity for Bubble Sort is O(1) (because only single additional memory space is required for temp variable). We can optimize bubble sort algorithm by stopping the algorithm if inner loop didn't cause any swap. Thus Bubble sort algorithm will take minimum time O(n)when elements in dataset are already sorted (best case). Even though Bubble sort is slow it has wide range of applications. For instance, in computer graphics it is popular for its capability to detect a very small error (like swap of just two elements) in almost-sorted arrays and fix it with just linear complexity (2n). Bubble Sort is also used in polygon filling algorithm, where bounding lines are sorted by their x coordinate at a specific scan line (a line parallel to x axis) and with incrementing y their order changes (two elements are swapped) only at intersections of two lines [7].

Enhancement of Bubble sort algorithm: One of the variations of bubble sort algorithm is Cocktail sort algorithm. It is also known as bidirectional bubble sort. It is both a stable algorithm and a comparison based sort. This algorithm differs from bubble sort in the fact that it sorts in both directions each pass through the dataset. The average number of comparisons is slightly reduced by this approach of Cocktail sort. This sorting algorithm is only marginally more difficult than Bubble Sort to implement. In case of best case we may conclude that Cocktail sort algorithm is the best algorithm to be used with regard to worst case analysis where the data elements are in reverse order it may be concluded that the proposed algorithm can be very effective for the small as well as large data sets. Another variation of Bubble sort algorithm is Comb sort. Comb sort is an improvement of bubble sort algorithm. The basic idea is to eliminate small values near the end of the dataset, since in a bubble sort these slow the sorting down tremendously [8].

B. Selection Sort

Selection sort is among the most intuitive sort of all the sorting algorithms. In this sort we find out the smallest elements in each pass and place it at the appropriate location. These steps are repeated until the dataset is sorted. In this method, to sort the data in increasing order, the first element is compared with all the elements in the dataset and if the first element is greater than the smallest element than the position of those two elements is swapped. So after the first pass, the smallest element is placed at the first position. The same procedure is repeated for every element until the dataset gets sorted.

The selection sort improves on the bubble sort by making only one exchange for every pass through the dataset. The advantage of selection sort algorithm is its easy implementation along with O(1) space complexity. Disadvantage of selection sort is its inefficiency for large datasets. You may see that the selection sort algorithm makes the same number of comparisons as the bubble sort algorithm and is therefore also O(n^2). However, due to the reduction in the number of exchanges, the selection sort

algorithm usually executes faster in benchmark studies. Selection sort has a quite important application because each item is actually moved at most once.

C. Insertion sort

Insertion sort is a simple sorting algorithm that is relatively efficient for small dataset and mostly for nearly sorted dataset. It is often is used as a part of more sophisticated algorithms. The insertion sort works like playing cards in which each card is placed at its proper place while playing in hands of a person. Sorting a hand of playing card is one of the real examples of insertion sort [11]. It works by taking elements from the dataset one by one and inserting them in their correct location into a new sorted dataset. In arrays, the new dataset and the remaining elements of the dataset can share the array's space, but insertion is costly as it requires shifting of all the following elements over by one. Advantages of Insertion sort is that it use O(1) auxiliary space. Generally, the Computational Complexity is O (n^2), but incase if the list is already sorted (best Case) complexity is O(n). It sorts the element of small dataset very quickly but in sorting the elements of large dataset it takes very long time. Insertion sort algorithm can take different running time to sort two input sequences of dataset of the same size depending upon how nearly they already sorted. Shell sort is a variant of insertion sort that is more efficient for larger dataset.

Enhancement of Insertion sort: An enhancement to the Insertion Sort algorithm can be in difference of approach as it compares with the very first element in the dataset, which in fact is the smallest element in the dataset at instant, after comparing (i)th element with (i-1)th . This is called as hit method; more we get hit more the efficiency increases. Basically sometimes we have element which gets sorted after (n-1) comparisons i.e at first place of dataset in insertion sort. So for reducing these useless comparison, why not we compare the element to be sorted with the very first element in the part of the dataset, which is already sorted i.e. before (i)th element, which we know is the smallest element up till now. Further list is divided, selecting a middle element and comparing to part on its left or right based on the condition for middle comparison and then comparing after leaving one element in that particular part, hence reducing the number of comparisons. The technique is more efficiently suitable for large dataset and efficiency increases when the (i)th is less than first element of dataset which gives O[n] in worst case. This work focuses to provide an enhancement in insertion sort and making enhanced insertion sort more efficient for larger datasets as it gives less than O(n^1.585) complexity in worst case and reduces near about half comparison. It does not requires scanning all elements, because of its hit method it provides a boost to sorting, also reduces the number of comparisons while sorting an array as compared to O(n^2) complexity of insertion sort, in fact it is O(n) in best as well as sometimes in average case. Basically the complexity of this algorithm varies from O(n) to O(n^1.585) [12].

D. Heap sort

Heapsort is a comparison-based sorting algorithm. Heapsort is an in-place algorithm, but is not a stable sort. Heap sort algorithm is a much more efficient version of selection sort algorithm. Like selection sort, it also works by finding the largest or the smallest element of the dataset and placing that at the end or the beginning of the dataset, then continuing with the rest of the dataset. The difference between the two is that Heap sort accomplishes this task efficiently by using the heap data structure. Heap is a special type of binary tree. Once the dataset has been converted into a heap, the root node is guaranteed to be the largest or the smallest element. When the root node is removed and placed at the end of the dataset, the heap is rearranged so the largest element remaining moves to the root. Using the heap data structure, finding the next largest element takes O(log n) time, instead of O(n) for a linear scan as in the case of selection sort. This permits Heap sort to run in O(n* log n) time.

The best part of using Heap sort is its worst case complexity which is O(n*logn) much better than Selection sort algorithm. The advantages of Heap sort is its time complexity and auxiliary space requirement which is O(1). Although Heap sort is somewhat slower in practice on most machines than a well-implemented quicksort, it has the advantage of a more favorable worst case O(n*log n) runtime. Thus, in-space and non-recursive makes it a good choice for large data sets. Heap sort has two major disadvantages that includes its slower speed than other such Divide and Conquer sorts that also have the same O (n*log n) time complexity due to cache behavior of Heap sort and other factors. The other disadvantage is Heap sort is unable to work when dealing with linked lists due to non-convertibility of linked lists to heap structure [5].

Enhanced Heap Sort: A new variant of Heap Sort is modified heap sort. Basic idea of this new Heap sort algorithm is similar to the classical Heap sort algorithm but it builds heap in a different way. This new algorithm requires nlogn-0.788928n comparisons for worst case and n*logn-n comparisons for average case. This algorithm uses only one comparison at each node. With one comparison we can decide which child of node contains larger element. This child is directly promoted to its parent position. In this way the algorithm walks down the path until a leaf is reached.

E. Quick Sort

Quick Sort is an in-place, massively recursive, divide and conquer sorting algorithm. It relies on a partition

operation to partition an array an element called a pivot is selected. All the elements that are smaller than the pivot are moved before it and all the greater elements are moved after it. This can be done efficiently in linear time and in-place. Efficient implementations of quick sort (with in-place partitioning) are typically unstable sorts and somewhat complex, but are among the fastest sorting algorithms in practice [2].

The most complicated issue in quick sort algorithm is choosing a good pivot element; consistently poor choices of pivots can result in drastically slower $O(n^2)$ performance, if at each step the median is chosen as the pivot then the algorithm works in $O(n^* \log n)$. Finding the median however, is an O(n) operation on unsorted dataset and therefore exacts its own penalty with sorting.

The quick sort is by far the fastest of the all the known sorting algorithms because Quicksort runs in O(n*log n) on the average case and in O(n^2) for the worst case. There are many versions of Quicksort algorithm. They are, using external memory, using in-place memory and using in-place memory with two pointers. The best part of Quicksort algorithm is its efficiency and fast processing. The disadvantage of Quicksort lies in the horrible results it shows for already sorted dataset.

F. Merge Sort

Merge sort was invented by John von Neumann and it belongs to the family of comparison-based sorting. Merge sort algorithm is very effective sorting technique when dataset is considerably large. It is an example of the divide-and conquer paradigm, that is, it breaks the data into two halves and then sorts the two half datasets recursively, and finally merges them to obtain the complete sorted list. The best part of Merge sort algorithm is that it has the time complexity of O(n*log n) for every case including worst, best and average case. Its worst-case running time has a lower order of growth than insertion sort.

Merge sort takes advantage of the ease of merging already sorted lists into a new sorted list. It starts by comparing every two elements and swapping them if the first element should come after the second element. It then merges each of the resulting dataset of two into datasets of four, and then merges those datasets of four, and so on; until at last two datasets are merged into the final dataset. Of the algorithms described here, this is the first algorithm that scales well to very large dataset, because its worst-case running time is O(n*log n).

Merge sort has seen a relatively recent escalation in popularity for practical implementations, as it is being used for the standard sort routine in the programming languages like Perl, Python and Java. The greatest advantage of merge sort is that it is well suited for large data set. Another strong point is that it is marginally faster than the heap sort for large datasets and it is often the best choice for sorting a linked list because the slow random access performance of a linked list makes it perform poorly for algorithms such as Quicksort and Heapsort. On the other hand, Merge sort can efficiently sort a linked list Disadvantage of merge sort is that it requires at least twice the memory requirements than other sorting algorithms because of its recursive nature. This is the biggest cause for concern in Merge sort as its space complexity is very high. Merge sort requires about a O(n) auxiliary space for its working.

G. Shell Sort

Shell sort is a generalization of Insertion sort, named after its inventor, Donald Shell. It is an improvisation of bubble sort and insertion sort by moving out of order elements more than one position at a time. One implementation can be described as arranging the data sequence in a two-dimensional array and then sorting the columns of the array using insertion sort [2]. It belongs to the family of in-place sorting algorithms but is regarded to be unstable.

Shell sort improves Insertion sort by comparing elements separated by a gap of several positions. Shell sort exploits the fact that insertion sort algorithm works efficiently on dataset that is already almost sorted. This algorithm is an example of an algorithm that is simple to code but difficult to analyze and understand theoretically. Although sorting algorithms exist that are more efficient, Shell sort remains a top and preferable choice for moderately large files and datasets because it has good running time and it is easy to code.

Enhancement of Shell Sort: An enhancement of Shell sort is Shaker sort. It is a variant of Shell sort that compares each adjacent pair of items in a dataset in turn, swapping them if necessary, and alternately passes through the list from the beginning to the end then from the end to the beginning. The algorithm stops when a pass does no swaps.

The complexity of the algorithm is O(n^2) for arbitrary data, but it approaches O(n) if the dataset is nearly in order at the beginning. Enhanced shell sort algorithm provides a powerful solution to decrease the number of comparisons as well as number of swaps to a minimum level in shortest possible time, thus decreasing the CPU execution time as well as saving the system memory. Enhanced shell sort algorithm offers least number of swaps on any size of data. The efficiency and working of this algorithm improves as the size of data grows.

IV. PERFORMANCE ANALYSIS

In order to compare and analyze the performance of the various Sorting algorithms above, we use a desktop computer (Intel Dual Core Processor @ 2.4GHz, 2GB

RAM, Windows 7 operating system) to do a serial experiments.

Under VS2008, using C language, the programs test the performances of various algorithms from input scale size by utilizing random function call and time function call [1]. The times taken by the CPU at execution for different inputs are shown in the table.

Results shows that for all small datasets the performance of all the techniques is almost same, but for the large datasets Quicksort is the most preferable algorithm followed by Shell sort and Merge sort algorithms.

TABLE I.

Sort Algorithms Time Cost Under Negative Input Sequence

Data size	Bubble	Insertion	Selection	Quick	Merge	Shell
1K	0.01	0.002	0.003	0.0003	0.02	0.0006
50K	12.7	2.824	4.064	0.0095	1.17	0.0004
70K	26.1	5.746	8.444	0.0136	1.64	0.0005
99K	0.82	11.01	15.71	0.0193	2.31	0.0683

V. CONCLUSION

Every sorting algorithm has some advantages and disadvantages. In the above analysis we had tried to show the strengths and weakness of popular and most commonly used sorting algorithms on the basis of their order, stability, memory usage, data type and computational complexity. To determine a good sorting algorithm, speed is the topmost priority but other factors like additional space requirements, handling various data types, length and complexity of code, worst-case behavior, caching, stability, and behavior on already-sorted or nearly-sorted data, consistency of performance cannot be ignored. In this paper, we got into sorting problem and investigated different solutions for comparison based sorting algorithms. From the discussion it can be deduced that every sorting algorithm can undergo a fine tuning with the intelligence aspects that are discussed so as to gain significant reduction in complexity values. We have also showed that the choice of sorting algorithm is not a straight forward topic, as a number of issues and factors must be considered as per the dataset.

REFERENCES

[1] D.T.V Dharmajee Rao , B.Ramesh ,"Experimental Based Selection of Best Sorting Algorithm", International Journal of Modern Engineering Research (IJMER) ,Vol.2, Issue.4, July-Aug 2012.

[2] https://en.wikipedia.org/wiki/Sorting_algorithm

[3] Pankaj Sareen ,"Comparison of Sorting Algorithms (On the Basis of Average Case)", International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE), Volume 3, Issue 3, March 2013.

[4] http://www.cprogramming.com/tutorial/

[5] Sonal Beniwal , Deepti Grover, "Comparison Of Various Sorting Algorithms: A review," International Journal of Emerging Research in Management &Technology ,Volume-2, Issue-5, May 2013.

[6] P. Dhivakar, G. Jayaprakash ,"Dual Sorting Algorithm Based on Quick Sort ," International Journal of Computer Science and Mobile Applications(IJCSMA), Vol.1 Issue. 6, December- 2013.

[7] http://geeksquiz.com/bubble-sort/

[8] V. Mansotra, Kr. Sourabh, "Implementing Bubble Sort Using a New Approach," Proceedings of the 5th National Conference; INDIACom-2011, March 10 – 11.

[9] Jariya Phongsai, Prof. Lawrence Muller, "Research Paper on Sorting Algorithms", http://www.eportfolio.lagcc.cuny.edu/,October 26, 2009.

[10] Basit Shahzad,Muhammad Tanvir Afzal, "Enhanced Shell Sorting Algorithm", International Journal of Computer, Electrical, Automation, Control and Information Engineering Vol:1, No:3, 2007.

[11] Ashutosh Bharadwaj, Shailendra Mishra, "Comparison of Sorting Algorithms based on Input Sequences", International Journal of Computer Applications (0975 – 8887), Volume 78 – No.14, September 2013.

[12] Tarundeep Singh Sodhi ,Surmeet Kaur , Snehdeep Kaur, "Enhanced Insertion Sort Algorithm",International Journal of Computer Applications (0975 – 8887),Volume 64– No.21, February 2013.